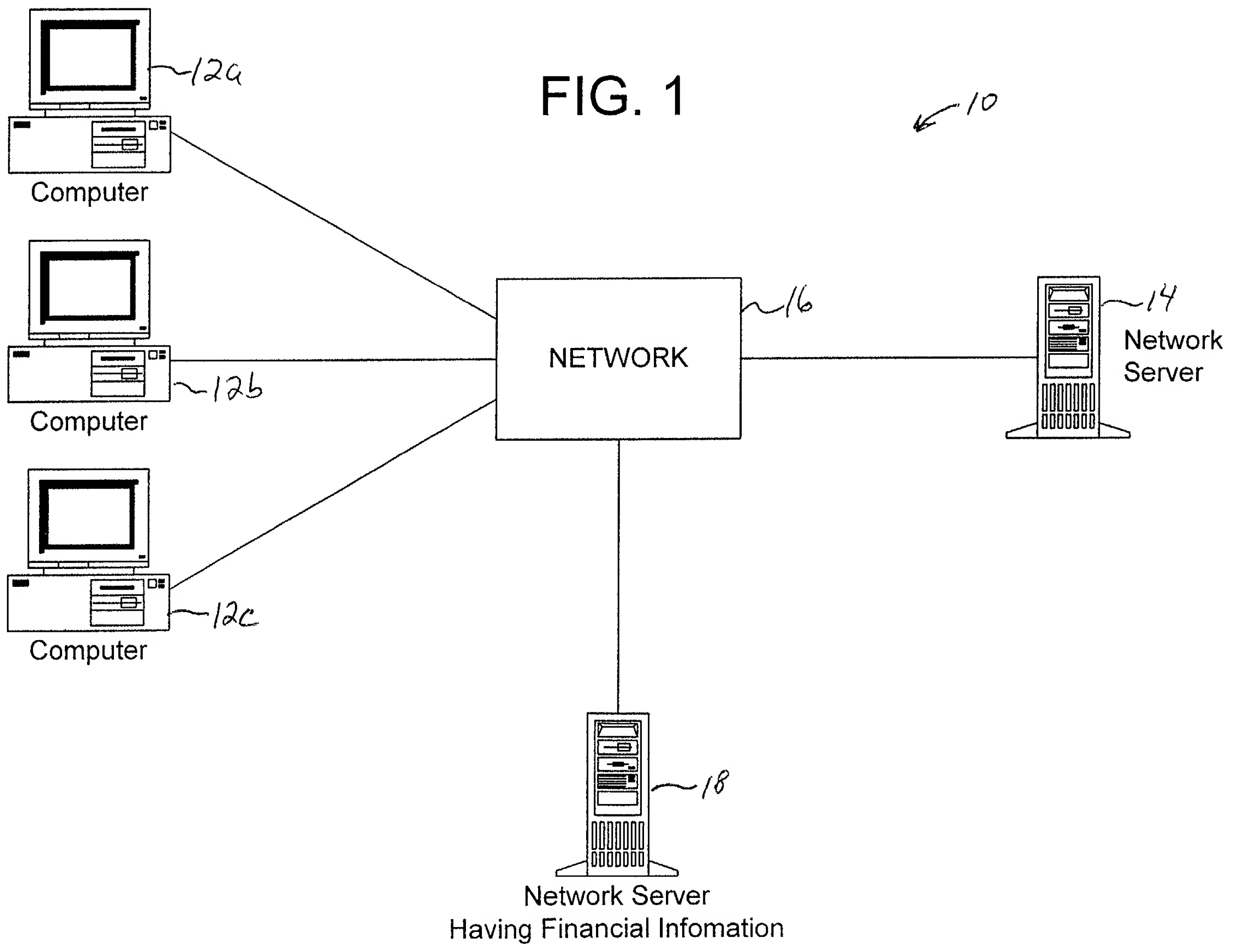
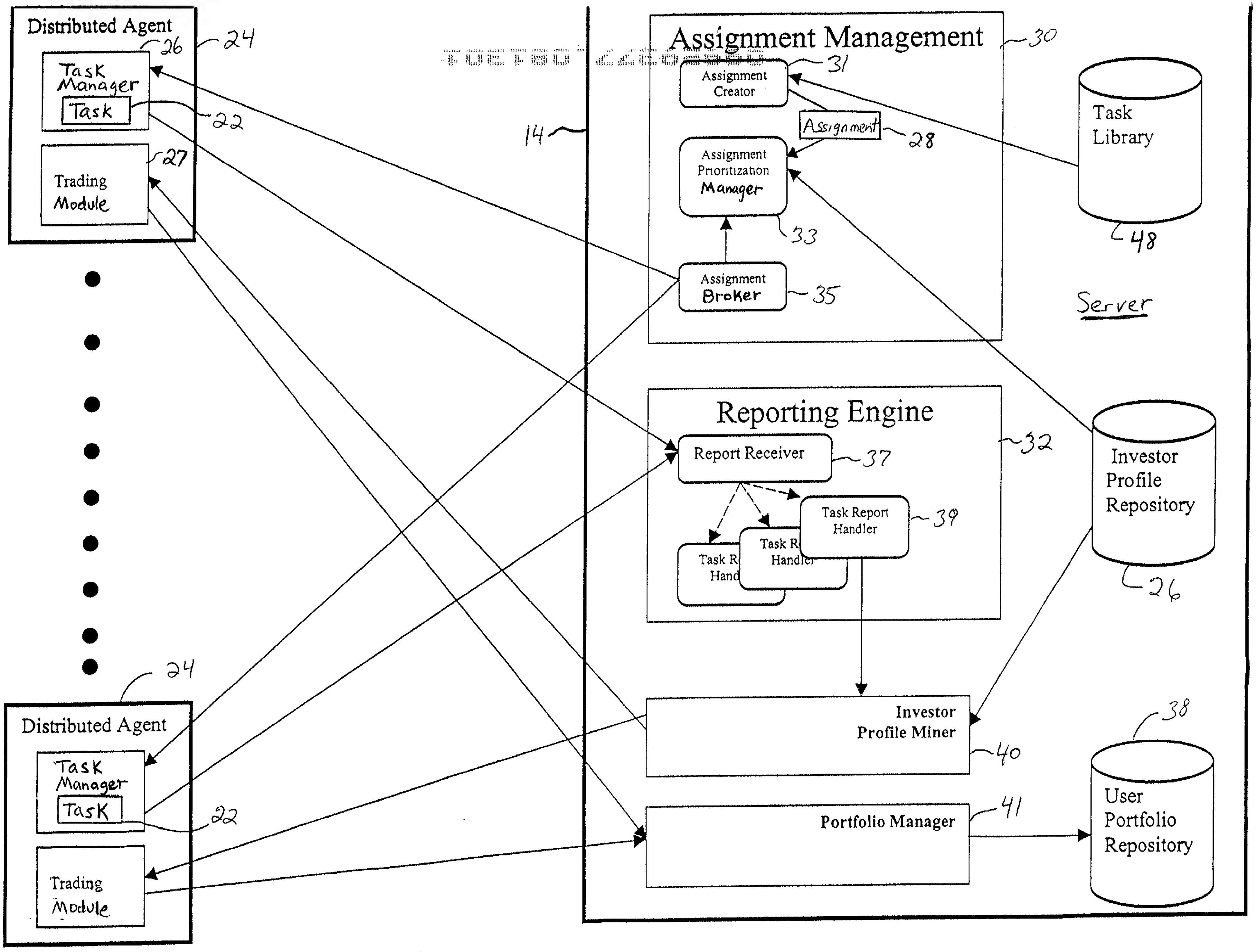
FIG. 1

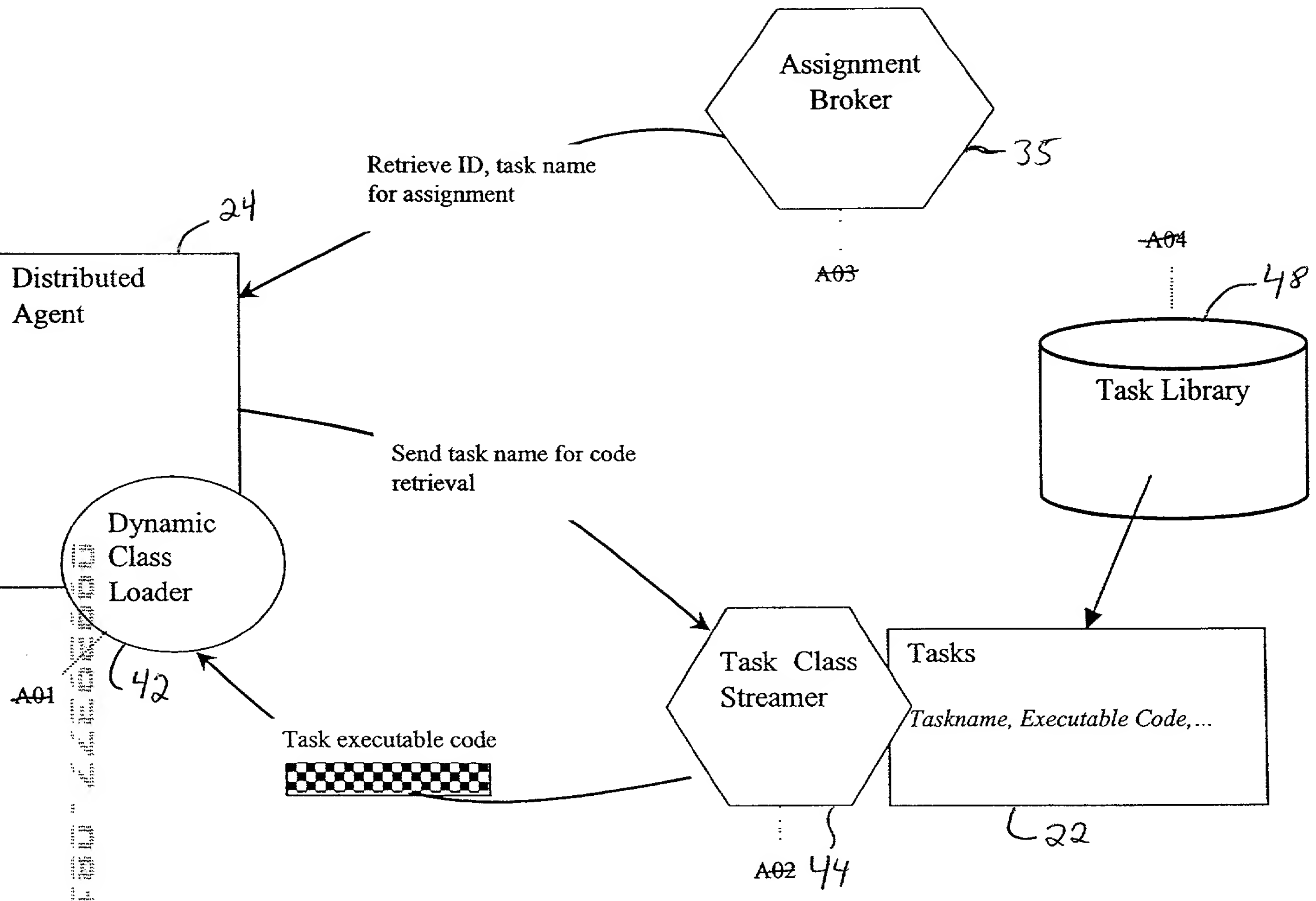


12a
Computer
12b
Computer
12c
Computer
NETWORK
16
14
Network Server
18
Network Server Having Financial Infomation
10



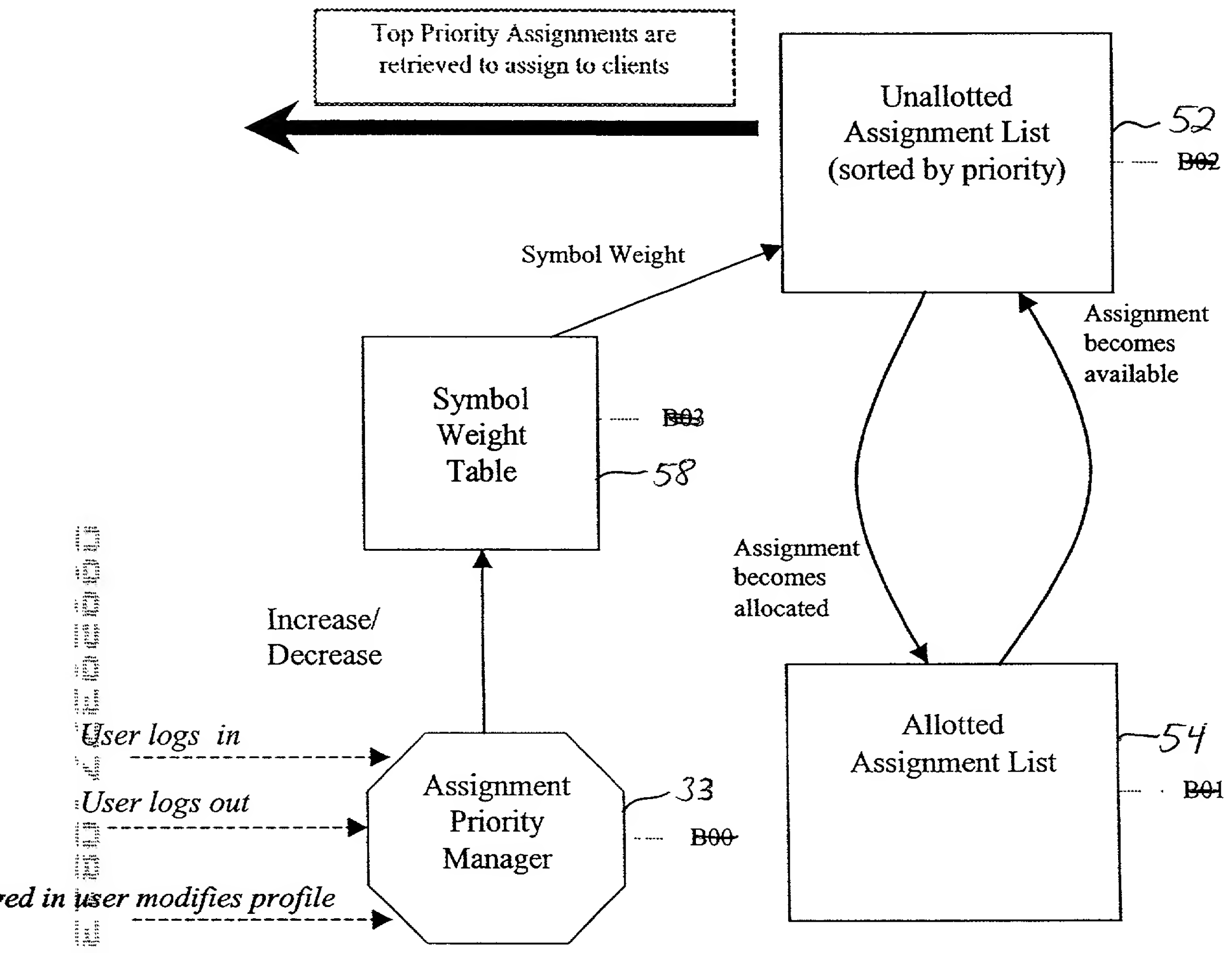
Distributed Agent
24
26
Task Manager
Task
22
Trading Module
27
14
Assignment Management
30
Assignment Creator
31
Assignment
28
Assignment Prioritization Manager
33
Assignment Broker
35
Task Library
48
Server
Reporting Engine
32
Report Receiver
37
Task Report Handler
39
Task R Hand
Task R Handler
Investor Profile Repository
26
Investor Profile Miner
40
Portfolio Manager
41
User Portfolio Repository
38
Distributed Agent
24
Task Manager
Task
22
Trading Module

FIG 2



Assignment Broker
35
Retrieve ID, task name for assignment
24
Distributed Agent
~~A03~~
~~A04~~
48
Task Library
Send task name for code retrieval
Dynamic Class Loader
Task Class Streamer
Tasks
Taskname, Executable Code,...
~~A01~~
42
Task executable code
~~A02~~
44
22

Fig. 3



Top Priority Assignments are retrieved to assign to clients
Unallotted Assignment List (sorted by priority)
52
~~B02~~
Symbol Weight
Symbol Weight Table
~~B03~~
58
Assignment becomes available
Assignment becomes allocated
Increase/ Decrease
Allotted Assignment List
54
~~B01~~
User logs in
User logs out
red in user modifies profile
Assignment Priority Manager
33
~~B00~~

Fig. 4

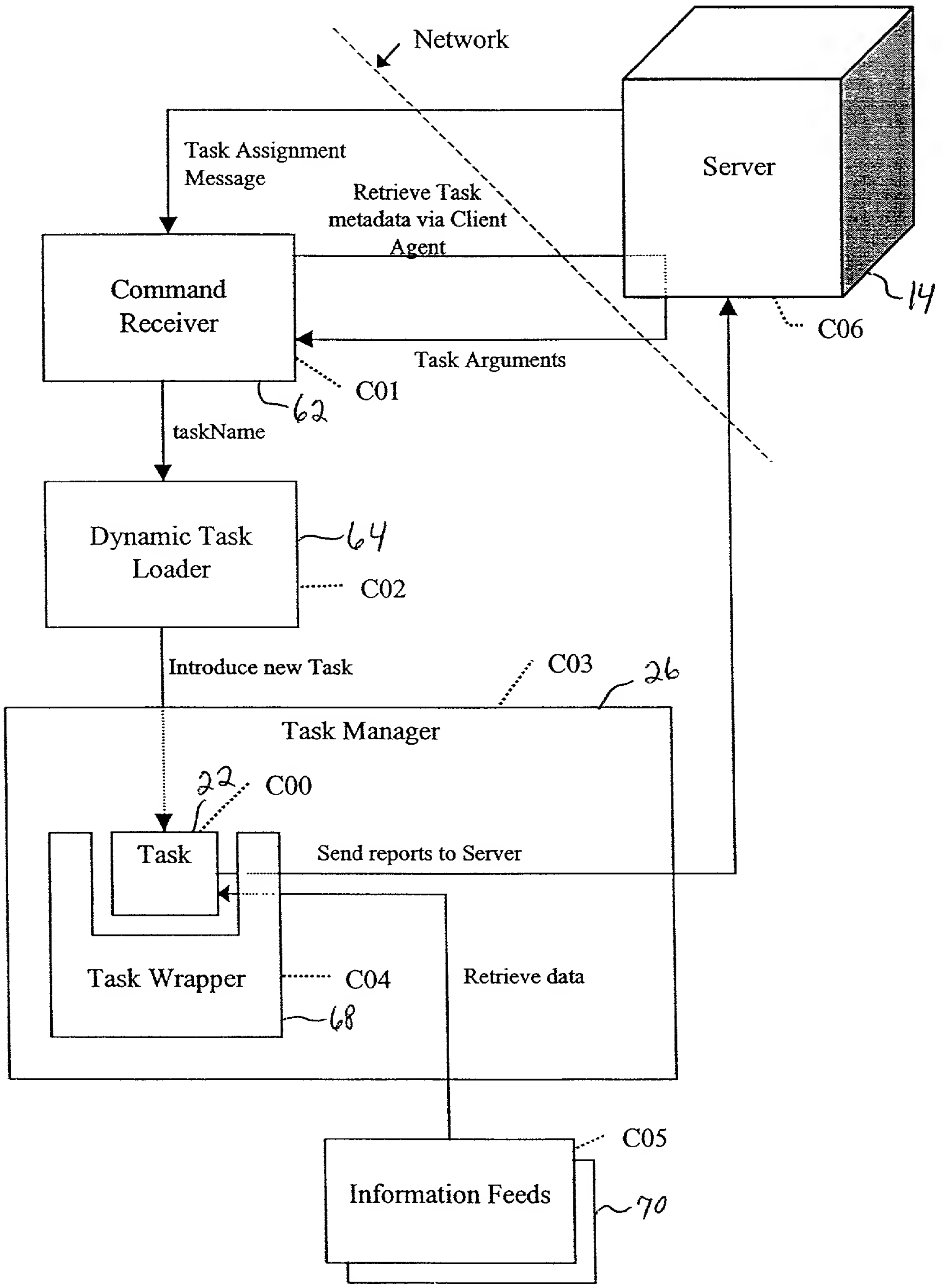
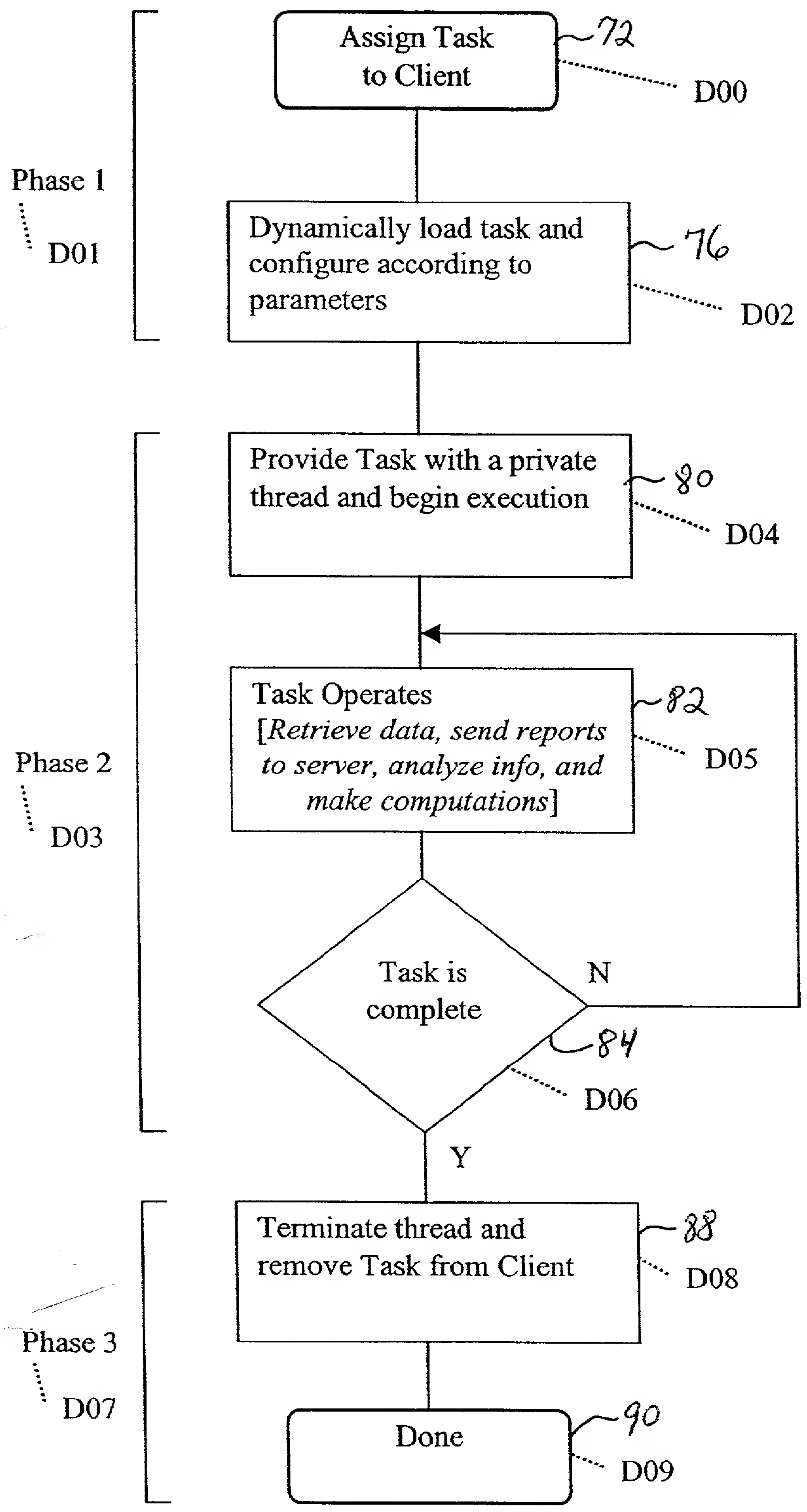

Network
Server
14
C06
Task Assignment Message
Retrieve Task metadata via Client Agent
Command Receiver
Task Arguments
C01
62
taskName
Dynamic Task Loader
64
C02
Introduce new Task
C03
26
Task Manager
22
C00
Task
Send reports to Server
Task Wrapper
C04
68
Retrieve data
C05
Information Feeds
70

Fig. 5



Assign Task to Client
72
D00
Phase 1
D01
Dynamically load task and configure according to parameters
76
D02
Provide Task with a private thread and begin execution
80
D04
Task Operates [*Retrieve data, send reports to server, analyze info, and make computations*]
82
D05
Phase 2
D03
Task is complete
N
84
D06
Y
Terminate thread and remove Task from Client
88
D08
Phase 3
D07
Done
90
D09

Fig. 6

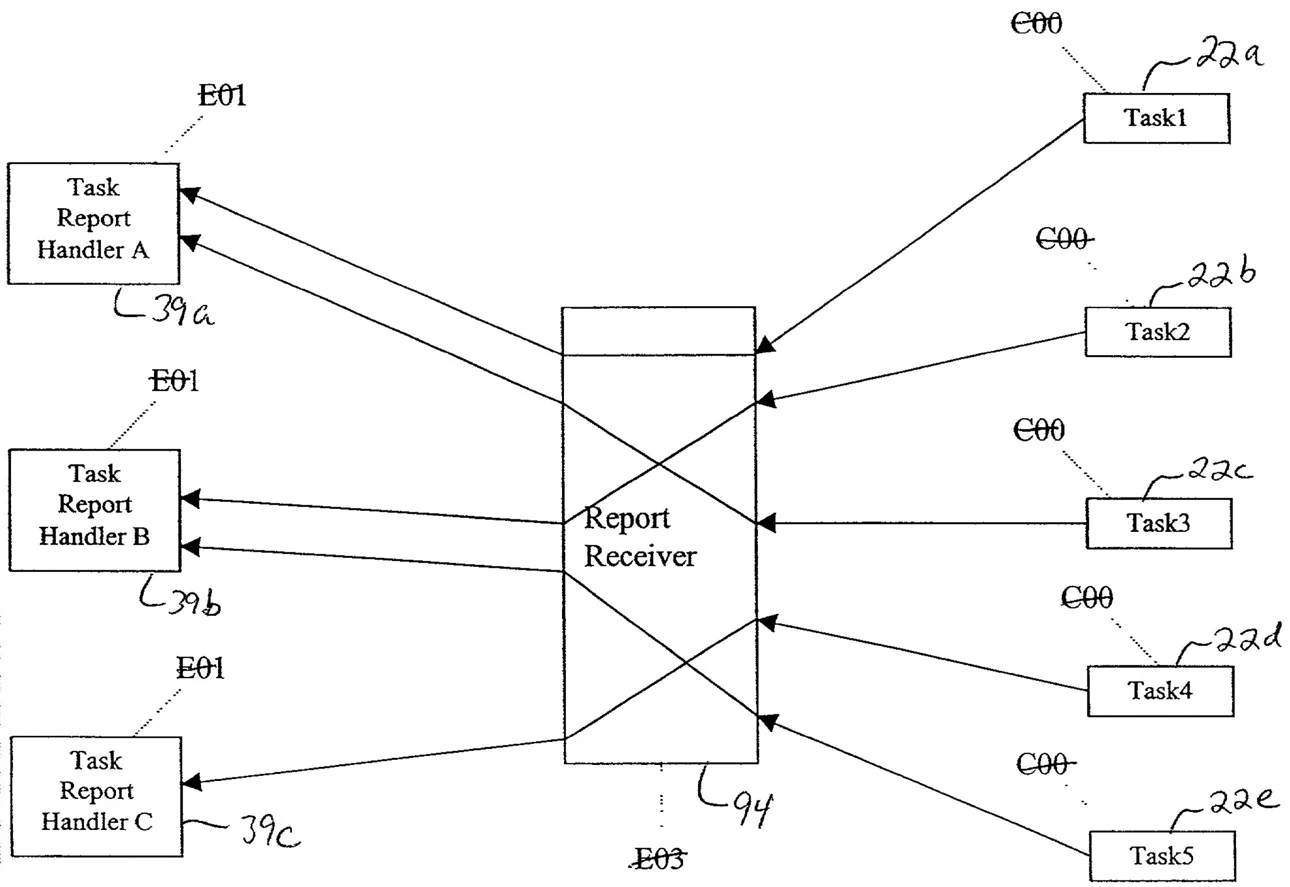
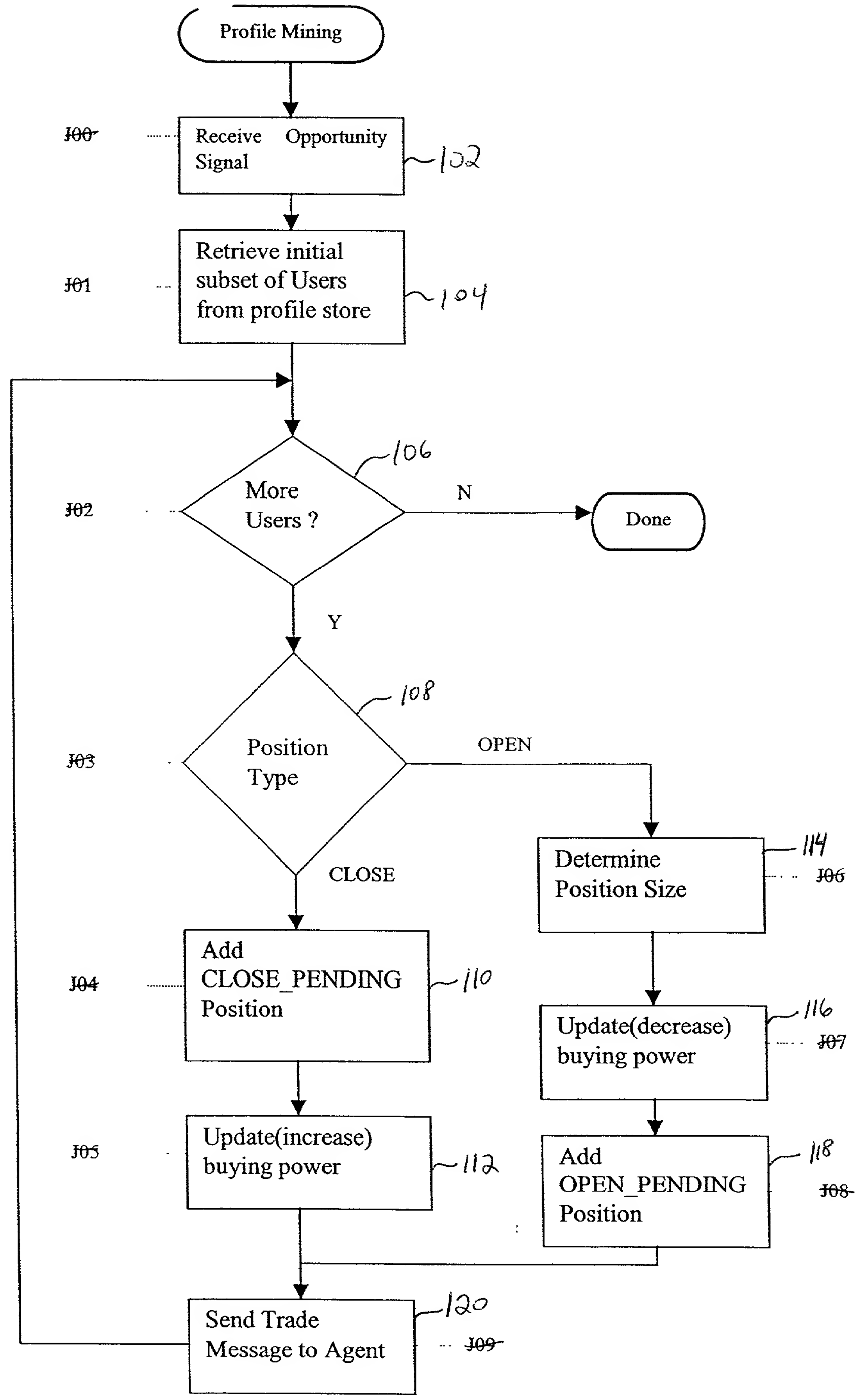
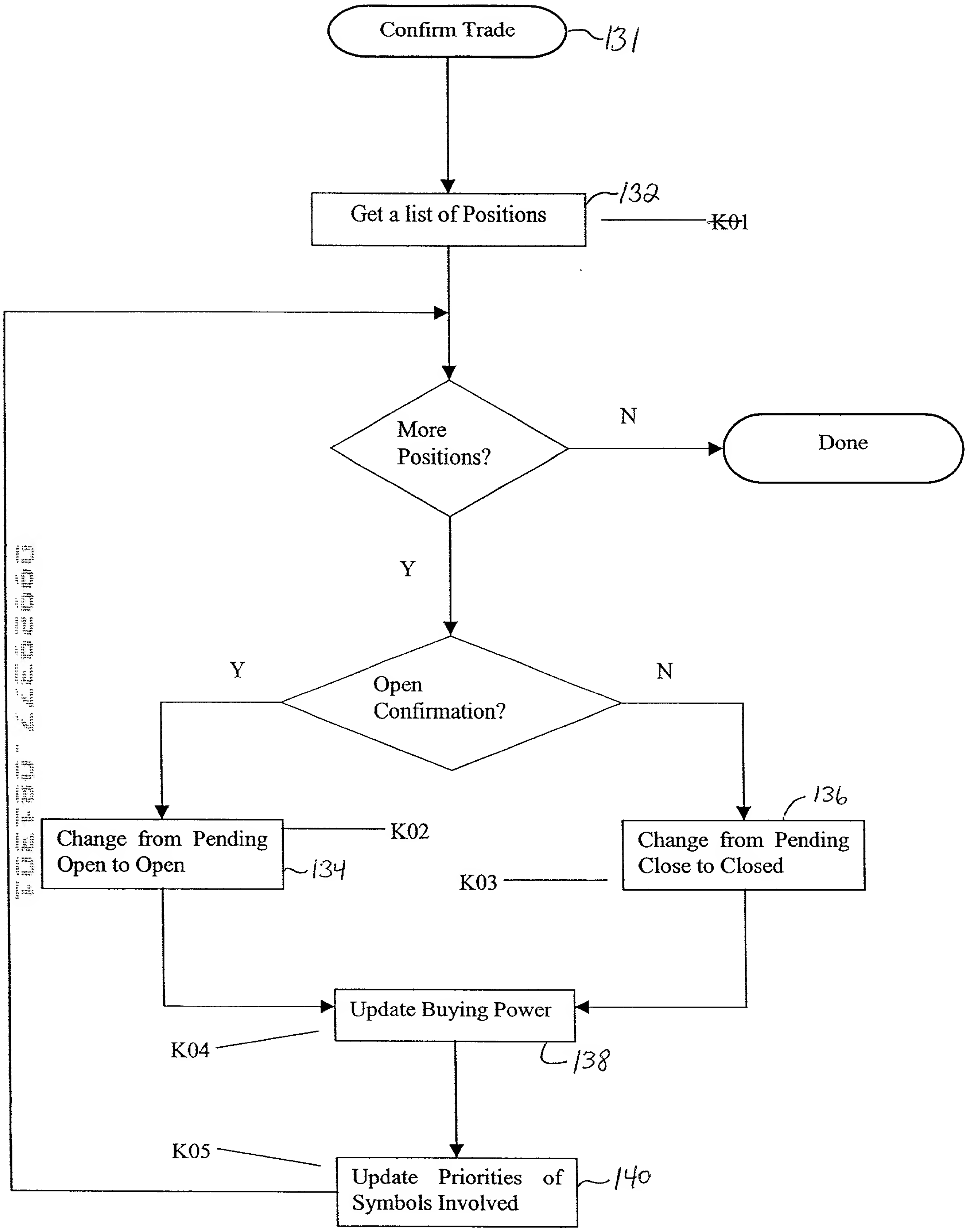

~~C00~~
22a
Task1
~~E01~~
Task Report Handler A
39a
~~C00~~
22b
Task2
~~E01~~
Task Report Handler B
39b
~~C00~~
22c
Task3
Report Receiver
~~C00~~
22d
Task4
~~E01~~
Task Report Handler C
39c
94
~~C00~~
22e
Task5
~~E03~~

Fig. 7



Profile Mining
~~100~~
Receive Opportunity Signal
102
~~101~~
Retrieve initial subset of Users from profile store
104
~~102~~
More Users ?
106
N
Done
Y
~~103~~
Position Type
108
OPEN
CLOSE
Determine Position Size
114
~~106~~
~~104~~
Add CLOSE_PENDING Position
110
Update(decrease) buying power
116
~~107~~
~~105~~
Update(increase) buying power
112
Add OPEN_PENDING Position
118
~~108~~
Send Trade Message to Agent
120
~~109~~

Fig. 8



Confirm Trade
131
Get a list of Positions
132
~~K01~~
More Positions?
N
Done
Y
Open Confirmation?
Y
N
Change from Pending Open to Open
K02
134
136
K03
Change from Pending Close to Closed
Update Buying Power
K04
138
K05
Update Priorities of Symbols Involved
140

Fig. 9

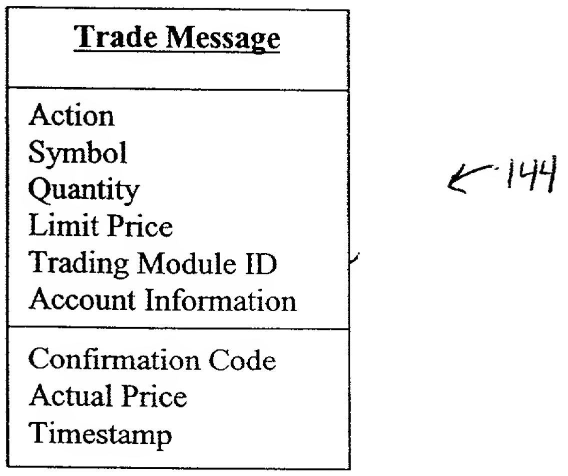
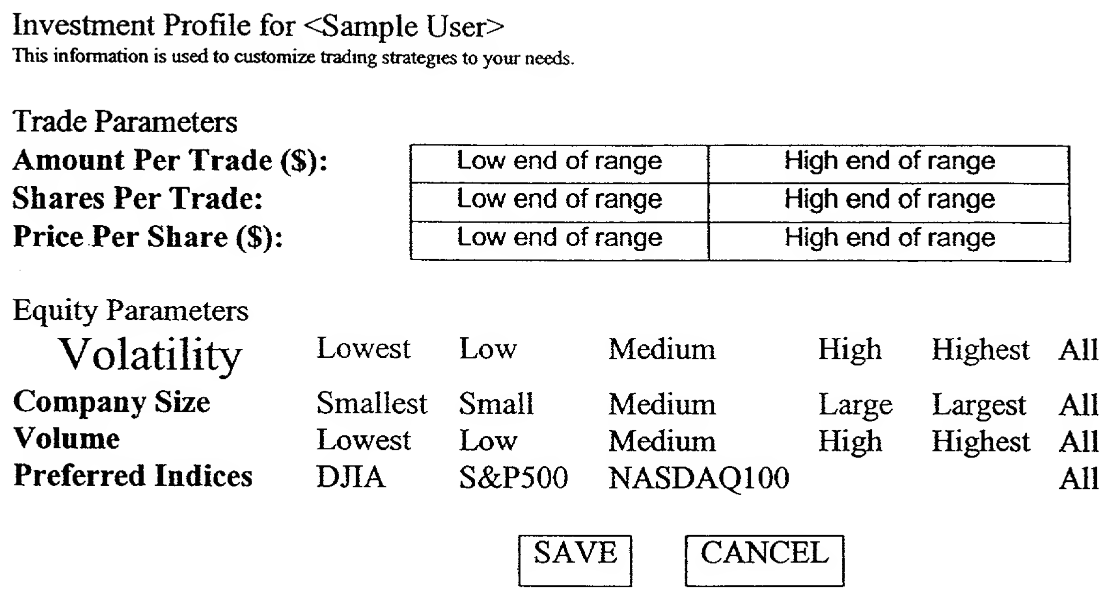

Trade Message
Action
Symbol
Quantity
Limit Price
Trading Module ID
Account Information
Confirmation Code
Actual Price
Timestamp
144

Fig. 10



Investment Profile for <Sample User>

This information is used to customize trading strategies to your needs.

Trade Parameters

Amount Per Trade ($):	Low end of range	High end of range
Shares Per Trade:	Low end of range	High end of range
Price Per Share ($):	Low end of range	High end of range

Equity Parameters

Volatility	Lowest	Low	Medium	High	Highest	All
Company Size	Smallest	Small	Medium	Large	Largest	All
Volume	Lowest	Low	Medium	High	Highest	All
Preferred Indices	DJIA	S&P500	NASDAQ100			All

Fig. 11

```
/**
 * Wrapper class to pass arguments to a task instance on the client.
 */
public class TaskArguments implements java.io.Serializable {

  /**
   * Unique name designating which class this task corresponds to.
   */
  public String taskName;

  /**
   * Execution parameters passed to a task when it is instantiated.
   * This usually takes the form of a Hash Table of objects.  The structure
   * is flexible to allow different numbers and sizes of
   * parameter to be passed to particular tasks.
   */
  public byte[] argByteArray;

}
```

Fig. 12A

```
/**
 * A collection of services that a task can utilize during its execution on the
 * client.  In order to maintain a high level of modularity, task communication
 * with either the client or server must occur through the methods of this
 * interface.
 */
public interface TaskServiceProvider extends Serializable {

  /**
   * Transmits a report to the server on the wrapped task's request.
   *
   * @param   reportText  the report to be sent
   */
  public void issueReport( String reportText );

  /**
   * Transmits a request for points to the server on the wrapped task's
request.
   */
  public void requestPoints();

  /**
   * Creates and installs a NewsDocReceiver for this task with the specified
feed.
   *
   * @param   task        the concerned task
   * @param   feedKey     describes the feed to use
   */
  public NewsDocReceiver installNewsDocReceiver( String feedKey );

  ///////////////////////////////////////////////////////////////////
  //Following are service request that tasks need during execution
  ///////////////////////////////////////////////////////////////////


  public Vector getQuotes(Vector symbols) throws SB_Exception;

  public void linkToDataFeed(Observer o, Vector symbols);
  public void unLinkFromDataFeed(Observer o, Vector symbols);

  public Vector getNASDAQTopVolumeLeaders(int num) throws SB_Exception;
  public Vector getNYSETopVolumeLeaders(int num) throws SB_Exception;
  public Vector getAMEXTopVolumeLeaders(int num) throws SB_Exception;

  public Vector getNASDAQTopPercentageLeaders(int num) throws SB_Exception;
  public Vector getNYSETopPercentageLeaders(int num) throws SB_Exception;
  public Vector getAMEXTopPercentageLeaders(int num) throws SB_Exception;

  public Vector  getHistoricalData(String symbol, Calendar startDay, Calendar
endDay) throws SB_Exception;
  public boolean checkIfMarketsOpen() throws SB_Exception;
}
```

Fig. 12B

```
/**
 * Provides access to the thread wait and notify methods.  This is used when an
 * object that is not the thread owner is running and wants wait/notify control
 * over its thread.
 */
public interface RemoteThreadMonitor {

  /**
   * Remote equivalent of Object.wait() .
   */
  public void remoteWait();

  /**
   * Remote equivalent of Object.notifyAll() .
   */
  public void remoteNotifyAll();

}
```

Fig. 12c